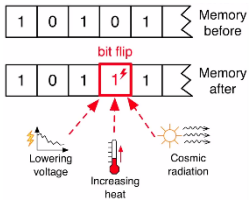
## *Algoritmos aleatorios.*

La NASA ha dicho que hay un hardware generador de números aleatorios en el Rover. Luego de eso, hicieron otro comentario, recordándonos que solo necesitamos que esos algoritmos **funcionen en la práctica**. Para que algo funcione en práctica, significa que hay siempre alguna **posibilidad de error**, pero tal vez la **probabilidad es tan pequeña** que la ignoramos en la practica, y si eso suena loco, solo tenemos que darnos cuenta de que en el mundo físico nada es incuestionable, siempre hay posibilidad de error.


Por ejemplo, el empaquetado de chips contiene pequeñas cantidades de contaminantes radioactivos, y cuando estos decaen, se liberan partículas alfa las cuales pueden en realidad cambiar bits en memoria, y quizás cambiar un número inesperadamente. Aún mas interesante, los rayos cósmicos pueden también causar pequeños errores, nunca podemos borrar la posibilidad de error completamente. ¿Pero cual es exactamente una probabilidad de error **aceptable** para la NASA?.

La NASA dice, "**Solo tenemos que estar seguros que, la probabilidad de error para una prueba dada, sea menor que la probabilidad de ganar la lotería dos veces seguidas**".

$$e = 6 \times 10^{-14}$$




Suficientemente seguro; no esperaremos ver un error, y esto podría correr cientos o miles de veces.
Ahora podemos preguntarnos, ¿el acceso a la aleatoriedad nos ayudaría a acelerar un algoritmo de decisión tal como esta prueba preliminar?

Por ejemplo, el empaquetado de chips contiene pequeñas cantidades de contaminantes radioactivos, y cuando estos decaen, se liberan partículas alfa las cuales pueden en realidad cambiar bits en memoria, y quizás cambiar un número inesperadamente. Aún mas interesante, los rayos cósmicos pueden también causar pequeños errores, nunca podemos borrar la posibilidad de error completamente. ¿Pero cual es exactamente una probabilidad de error **aceptable** para la NASA?.

Un algoritmo aleatorio (o probabilístico, o no determinista) es un algoritmo que basa su resultado en la toma de algunas decisiones al azar, de tal forma que, en promedio, obtiene una buena solución al problema planteado para cualquier distribución de los datos de entrada. Es decir, al contrario que un algoritmo determinista, a partir de unos mismos datos se pueden obtener distintas soluciones y, en algunos casos, soluciones erróneas.

- Uso de probabilidades en el análisis del algoritmo
- Usa generadores de números aleatorios

- La “aleatoriedad” es la característica clave
- Usado para problemas NP-Hard o NP-Completos

Ejemplo de algoritmo aleatorio:

```
findA(array A, n)
begin
    repeat
        randomly select one element out of n elements.
    until ‘X’ is found.
end
```

Existen varios tipos de algoritmos probabilísticos dependiendo de su funcionamiento, pudiéndose distinguir:

**Algoritmos numéricos**, que proporcionan una solución aproximada del problema.

**Algoritmos de Montecarlo**, que pueden dar la respuesta correcta o respuesta erróneas (con probabilidad baja). El algoritmo Montecarlo falla con cierta probabilidad, pero no puede decir cuando falla. Podemos reducir la probabilidad de falla significativamente, por medio de la ejecución repetitiva pudiendo tomar la mayoría de las respuestas produce un si o un no.

**Algoritmos de Las Vegas**, que nunca dan una respuesta incorrecta: o bien no encuentran la respuesta correcta e informan del fallo. El algoritmo de Las vegas falla con cierta probabilidad, pero nos avisa cuando falla. Podemos correrlo una y otra vez hasta que tenga éxito. En otras palabras, Las vegas es un algoritmo que corre por un tiempo indeterminado y no predecible pero siempre tiene éxito.

## *Algoritmos aleatorios (Montecarlo).*

Bajo el nombre de ***“Método de Monte Carlo”*** se agrupan una serie de procedimientos que analizan distribuciones de variables aleatorias usando simulación de números aleatorios. El Método de Monte Carlo da solución a una gran variedad de problemas matemáticos haciendo experimentos con muestreos estadísticos en una comput adora. El método es aplicable a cualquier tipo de problema, ya sea ***estocástico o determinístico***.

Generalmente en Estadística los modelos aleatorios se usan para simular fenómenos que poseen algún componente aleatorio, pero en el método de Monte Carlo, el objeto de la investigación es el objeto en sí mismo, un suceso aleatorio o pseudo-aleatorio se usa para estudiar el modelo.

A veces la aplicación del método de Monte Carlo se usa para analizar problemas que no tienen un componente aleatorio explícito; en estos casos un parámetro determinista del problema se expresa como una distribución aleatoria y se simula dicha distribución.

***La simulación de Monte Carlo fue creada para resolver integrales que no se pueden resolver por métodos analíticos***, para resolver estas integrales se usaron números aleatorios.

El nombre y el desarrollo sistemático de los métodos de Monte Carlo data aproximadamente de 1944 con el desarrollo de la computadora electrónica. Sin embargo, hay varias instancias (aisladas y no desarrolladas) en muchas ocasiones anteriores a 1944.
El uso real de los métodos de Monte Carlo como una herramienta de investigación, viene del trabajo de la bomba atómica durante la Segunda Guerra Mundial.

Variantes del Método de Monte Carlo se usan para resolver problemas muy diversos. De todas ellas, en el caso de los cálculos computacionales relacionados con sistemas moleculares, las más importantes son las siguientes:
**1-Método Clásico (Classical Monte Carlo, CMC)**: aplicación de distribuciones de probabilidades (generalmente la distribución clásica de Maxwell y Boltzmann) para obtener propiedades termodinámicas, estructuras de energía mínima y constantes cinéticas.
**2-Método Cuántico (Quantum Monte Carlo, QMC)**: uso de trayectorias aleatorias para calcular funciones de onda y energías de sistemas cuánticos y para calcular estructuras electrónicas usando como punto de partida la ecuación de Schroedinger.
**3-Método de la Integral a lo largo de la Trayectoria (Path-Integral Quantum Monte Carlo, PIMC)**: cálculo de las integrales de la Mecánica Estadística Cuántica para obtener propiedades termodinámicas y constantes cinéticas usando como punto de partida la integral a lo largo de la trayectoria de Feynman.
**4-Método Volumétrico (Volumetric Monte Carlo, VMC)**: uso de números aleatorios y cuasi-aleatorios para generar volúmenes moleculares y muestras del espacio de fase molecular).
**5-Método de Simulación (Simulation Monte Carlo, SMC)**: uso de algoritmos aleatorios para generar las condiciones iniciales de la simulación de trayectorias cuasi-clásicas o para introducir efectos estocásticos ("termalización de las trayectorias") en Dinámica Molecular. (El así llamado "Método Cinético" —Kinetic Monte Carlo, KMC— es uno de los SMC.).

## *Algoritmos aleatorios (Las Vegas).*

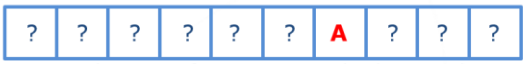

Un algoritmo tipo Las Vegas es un algoritmo de computación de carácter aleatorio (random) que no es aproximado: es decir, da el resultado correcto o informa que ha fallado.
Un algoritmo de este tipo no especula con el resultado sino que especula con los recursos a utilizar en su computación.

De la misma manera que el método de Montecarlo, la probabilidad de encontrar una solución correcta aumenta con el tiempo empleado en obtenerla y el número de muestreos utilizado. Un algoritmo tipo Las Vegas se utiliza sobre todo en problemas NP-completos, que serían intratables con métodos determinísticos.
Existe un riesgo de no encontrar solución debido a que se hacen elecciones de rutas aleatorias que pueden no llevar a ningún sitio. El objetivo es minimizar la probabilidad de no encontrar la solución, tomando decisiones aleatorias con inteligencia, pero minimizando también el tiempo de ejecución al aplicarse sobre el espacio de información aleatoria.

La clase de complejidad de los problemas de decisión de estos algoritmos con ejecución polinómica es ZPP.
ZPP=RP*no-RP

Su esquema de implementación se parece al de los algoritmos de Montecarlo, pero se diferencian de ellos en que incluyen una variable booleana para saber si se ha encontrado la solución correcta.

## *Algoritmos aleatorios (Las Vegas vs Montecarlo).*



Arreglo para verificar si tiene un valor determinado

```
Function LasVegasSearch(A array; n char; size:integer);
Begin
        result=false;
        While (result<>true) do
        Begin
                pos=random(size);
                if (n=A[pos]) then
                        result=true;
        end
        return result;
end
```

```
Function MonteCarloSearch(A array; n char; size, x:integer);
Begin
        i=0;
        result=false;
        While (i <= x) do
        Begin
                i=i+1;
                pos=random(size);
                if (n=A[pos]) then
                        result=true;
        end
        return result;
end
```

*Una mejora que se le puede hacer es cortar cuando lo encuentre*

## Test Aleatorio de Primalidad

Un **test de primalidad** (o chequeo de primalidad) es un algoritmo que, dado un número de entrada n, no consigue verificar la hipótesis de un teorema cuya conclusión es que n es compuesto. Esto es, un test de primalidad sólo conjetura que ***"ante la falta de certificación sobre la hipótesis de que n es compuesto podemos tener cierta confianza en que se trata de un número primo".***
Esta definición supone un grado menor de confianza que lo que se denomina **prueba de primalidad** *(o test verdadero de primalidad)*, que ofrece una seguridad matemática al respecto.


Ahora bien, ***¿cuál es nuestra necesidad o importancia de corroborar si un número es o no un número primo?***

Su importancia radica, por ejemplo, en que actualmente los métodos criptográficos usan números primos de muchas cifras decimales (*llamados primos industriales*) como parte fundamental del proceso de encriptación.
El mayor problema es que la seguridad del método se diluye cuando elegimos un número que creemos es primo cuando sin embargo no lo es, por lo que es fundamental tener algoritmos rápidos y eficientes que certifiquen la primalidad.

**Sea n ∈ N. n es primo si y sólo si los únicos divisores que posee son 1 y n**

Esta definición se generaliza para los dominios de integridad, lo cual da lugar al concepto de elemento primo. Otra definición podría ser:
***Sea D un dominio de integridad y sea n ∈ D. n es un elemento primo si y sólo si:***

***a) n = 0***
***b) n no es una unidad.***
***c) Si p divide a ab con a, b ∈ D entonces p|a o bien p|b.***

Ahora que tenemos unas definiciones debemos poder certificar si un número es primo o no.
Uno de los primeros algoritmos para probar la primalidad es el que indica que:
Como input toma n ∈ Z

Paso. 1 Escribir todos los números hasta n.
Paso. 2 Tachar el 1 pues es una unidad.
Paso. 3 Repetir hasta n

Tachar los múltiplos del siguiente número sin tachar, excepto el mismo número.
Output = Los números tachados son compuestos y los no tachados son primos.

Este algoritmo tiene la particularidad de que no sólo nos certifica si un número es primo o no, sino que nos da todos los primos menores que él.
El algoritmo funciona pues estamos comprobando si es o no múltiplo de algún número menor que el, lo cual es equivalente a la definición de número primo.
*Este método es óptimo cuando necesitamos construir la tabla con todos los números primos hasta n, pero* ***es tremendamente ineficiente para nuestro propósito****, pues se trata de un algoritmo exponencial tanto en tiempo como en espacio.*

**División por tentativa - Trial division *(o test de divisibilidad)***: Consiste en estudiar la divisibilidad de un número impar y se toma el hecho de que ningún numero mayor a 3 es primo si no es divisible entre ningún otro impar menor o igual que $\sqrt{N}$.
Es un algoritmo sencillo para comprobar la primalidad de un número. Es muy rápido, pero no sirve para números grandes*(64 bits)*, solo para números pequeños *(32 bits)*. Ya que se necesita comprobar cerca de $\sqrt{N}/_2$ divisores, empleándose una cantidad de tiempo del orden de $O(\sqrt{N})$.
La división por tentativa garantiza encontrar un factor de *n*, puesto que comprueba todos los factores primos posibles de *n*. Por tanto, si el algoritmo no encuentra ningún factor, es una prueba de que *n* es primo.

**Pequeño teorema de Fermat**: El pequeño teorema de Fermat da una condición necesaria para que un número p sea primo.

Si p es un número primo y $0 < A < P$ entonces, $A^{P-1} \equiv 1(mod\ P)$.
Si consideramos un k cualquiera verificado $1 < k < P$. Como P es primo y menor que A y k, $Ak \equiv 0(mod\ P)$ es imposible. Conisderemos cualquier $1 \leq i \leq j < P$. $Ai \equiv Aj(mod\ P)$ implicaria $A(j-i) \equiv 0(mod\ P)$ pero esto es imposible dado el argumento anterior, ya que $1 \leq j - i < P$.

***Si el recíproco del Teorema pequeño de Fermat fuese cierto***, entonces tendríamos un test de primalidad computacionalmente equivalente a la exponenciación modular (esto es, $O(\log N)$). Desafortunadamente ***el resultado recíproco no es cierto***.

Otro algoritmo elemental es el de divisiones sucesivas o de **Fibonacci**:
Input = n ∈ Z
Paso. 1 Mientras i < √n
Si n = 0 mod i entonces Output = Compuesto
Paso. 2 Output = Primo
Este algoritmo es exponencial en tiempo
Ambos algoritmos fueron de los primeros utilizados para comprobar la primalidad.

Avanzando en el tiempo podemos mencionar el Test de **Miller-Rabin**:
Input = n, k ∈ Z /* k = número máximo de repeticiones */
Paso. 1 Escribir n − 1 = 2sd dividiendo sucesivas veces n − 1 entre 2.
Paso. 2 Desde i = 1 hasta k hacer
Sea a un número escogido aleatoriamente en {1, . . . , n − 1}
Si ad = 1 mod n entonces Output = Compuesto.
Desde r = 0 hasta s − 1 hacer
• Si a2rd = −1 mod n entonces Output = Compuesto.
Paso. 3 Output = Probable Primo.
Este algoritmo es del orden O(log2 (n))
Ahora veremos el algoritmo conocido como **AKS**, el cual distingue entre números primos y compuestos
Input = n ∈ Z
Paso. 1 Si n = ab para algún a ∈ N y b > 1 entonces Output = Compuesto.
Paso. 2 Encuentra el menor r tal que or(n) > 4 log2(n)
Paso. 3 Si 1 < (a, n) < n para algún a =< r entonces Output = Compuesto.
Paso. 4 Si n =< r entonces Output = Primo.
Paso. 5 Desde a = 1 hasta 2√p(r) log(n) comprueba
si (x + a)n = xn + a mod (xr − 1, n) entonces Output = Compuesto.
Paso. 6 Output = Primo

Los autores demostraron además que, si determinados números primos *(llamados números primos de Sophie Germain)* tienen la distribución conjeturada por el matemático austriaco Emil Artin, el exponente

21/2 que aparece en la expresión de complejidad puede reducirse a 15/2. Lo que implica que el tiempo estimado de ejecución sería equivalente a O((log n)6+e) de forma incondicional.
En realidad este descubrimiento no tiene implicaciones prácticas en la computación moderna. Lo cierto es que las partes constantes de la complejidad del algoritmo son mucho más costosas que en los actuales algoritmos probabilísticos. Es de esperar que en el futuro cercano se obtengan mejoras en esas constantes, pero lo cierto es que los algoritmos actuales de generación de números primos cubren bastante bien las necesidades actuales y, posiblemente, las futuras (y es poco probable que la línea propuesta mejore en tiempo de ejecución a los algoritmos probabilísticos existentes). Sin embargo sí que tiene una importancia fundamental desde el punto de vista teórico, ya que supone la primera prueba de primalidad de estas características que ha sido matemáticamente demostrada.